Amendments to the Claims:

This listing of claims will replace all prior version, and listings, of claims in the application.

Listing of Claims:

1 (Cancelled).

2 (Cancelled).

3 (Cancelled).

4 (Cancelled).

5 (Cancelled).

6 (Cancelled).

7 (Cancelled).

8 (Cancelled).

9 (Cancelled).

10 (Cancelled).

11 (Cancelled).

12 (Cancelled).

13 (Cancelled).

14 (Cancelled).

15 (Canceled)

16 (Currently Amended) A method for obfuscation of computer program execution flow to increase computer program security, the method comprising-:

(a) ~~breaking the computer code into a plurality of code segments, and identifying~~ partitioning the computer program into one or more non-critical code segments and one or more critical code segments to be obfuscated;

(b) removing at least one of the critical code segments and embedding ~~each of one or more~~ the critical code segments within ~~respective first~~ one or more exception handlers~~, wherein when the computer program is executed, the critical code segments are executed within the respective first exception handlers in a kernel-level unprotected mode, wherein an execution path of the critical code segments is hidden from a debugger program executing in a user-level protected mode~~;

(c) providing the computer program with an exception set-up handler to set up operation of the ~~first~~ exception handlers during the execution of the computer program;

(d) providing the computer program with ~~embedding~~ an in-line code segment inserted within ~~a first~~ one of the ~~remaining plurality of~~non-critical code segments for setting up and invoking the exception set-up handler~~,~~; and

~~(f)~~(e) executing the computer program on a computer having an operating system that provides kernel-level and user-level execution modes, and debug resources to support generation and processing of exceptions at specified addresses, wherein ~~when the computer program is executed,~~ execution flow of computer program is maintained with the ~~remaining plurality of~~non-critical code segments ~~are executed~~ executing in the user-level protected mode,~~ wherein an execution path of the remaining plurality of code segments is visible to the debugger program executing in the user-level protected mode~~ but when an address immediately prior to each one of the critical code segments in the execution flow is encountered, an exception occurs and the kernel-level execution mode is entered whereby control is transferred to the one or more exception handlers for execution of the critical code segments, and when the exception handlers complete execution, control is returned to non-critical code segments.

17 (Currently Amended) The method of claim 16 wherein the providing (c) further includes: using the set-up handler to initiate a set-up of debug registers, such that the critical code segments in the exception handlers are executed at appropriate times during ~~the program~~ execution of the computer program.

18 (Previously Presented) The method of claim 16 wherein the embedding (d) further includes: executing the in-line code segment prior to a point in the program flow where any of the critical code segments was removed for placement into the exception handlers.

19 (Currently Amended) The method of claim 18 further including: using the in-line code segment to install the exception set-up handler on an exception handler linked list for a current thread, such that when~~ an~~ one of the exceptions occurs, the operating system hands control to the exception set-up handler for execution.

20 (Previously Presented) The method of claim 19 further including: removing the exception set-up handler from the linked list after execution.

21 (Previously Presented) The method of claim 17 wherein the embedding (b) further includes: inserting each of the exception handlers into the linked list.

22 (Currently Amended) The method of claim 21 further including: when ~~an~~ one of the exceptions is raised to the operating system, handing control down the linked list until one of the exception handlers determines that the exception is one that the exception handler is designed to handle.

23 (Previously Presented) The method of claim 22 further including: if the exception is one for which the current exception handler was designed to handle, executing the critical code segment included in the current exception handler.

24 (Previously Presented) The method of claim 23 further including: if the exception is one for which the exception set-up handler was designed to handle, setting a return address for the exception set-up handler when the exception processing completes.

25 (Previously Presented) The method of claim 24 further including: modifying debug registers during execution of the exception handlers such that any number of exception handlers may be daisy chained.

26 (Currently Amended) The method of claim 16 wherein the embedding (b) further includes providing entry code for the ~~first~~ exception handlers to determine a nature of the exception.

27 (Currently Amended) The method of claim 26 further including handling the exception if the nature of the exception is applicable to ~~the~~ a first exception handler, otherwise handing exception processing to the next available exception handler.

28 (Previously Presented) The method of claim 16 further including obfuscating the critical code segments to prevent reverse engineering.

29 (Previously Presented) The method of claim 16 further including obfuscating the critical code segments to hide anti-piracy algorithms.

30 (Currently Amended) The method of claim 16 further including providing code for the ~~first~~ exception handlers to modify a return address to be used after completion of the exception processing.

31 (Previously Presented) The method of claim 30 further including rearranging non-embedded code segments out of normal execution order, and setting exception addresses and return addresses to ensure proper program sequencing to further obfuscate the program execution flow.

32 (Currently Amended) The method of claim 16 further including code within the ~~first~~ exception handlers to modify ~~the~~ exception conditions to support future exceptions.

33 (Currently Amended) The method of claim 32, further including embedding a plurality of critical code segments within at least one of the ~~first~~ exception handlers.

34 (Currently Amended) The method of claim 33, further including selecting which of a plurality of critical code segments within the at least one ~~first~~ exception handler should be executed based on ~~the~~ exception information.

35 (Currently Amended) The method of claim 32, further including embedding one or more ~~first~~ exception handlers into other ~~first~~ exception handlers to further obfuscate the program execution flow.

36 (Currently Amended) A method for obfuscation of computer program execution flow to increase computer program security, the method comprising :

(a) partitioning the program into ~~a plurality of~~one or more non-critical code segments and ~~at least one~~one or more critical code segments;

(b) obfuscating at least one of the critical code segments by removing the critical code segments and embedding each of one of the critical code segments within ~~a respective~~one or more exception handlers~~, wherein when the computer program is executed, the critical code segments are executed within the respective first exception handlers in a kernel-level unprotected mode, wherein an execution path of the critical code segments is hidden from a debugger program executing in a user-level protected mode~~;

(c) providing the computer program with a driver to set up operation of the ~~first~~ exception handlers during execution of the computer program; and

(d) providing the computer program with ~~embedding~~ an in-line code segment inserted within a ~~first~~ non-critical code segment for invoking the driver when the program is executed~~,~~; and

(e) executing the computer program on a computer having an operating system that provides kernel-level and user-level execution modes, and debug resources to support generation and processing of exceptions at specified addresses, wherein ~~when the computer program is executed,~~ execution flow of computer program is maintained with the ~~remaining plurality of~~non-critical code segments ~~are executed~~ executing in the user-level protected mode,~~ wherein an execution path of the remaining plurality of code segments is visible to the debugger program executing in the user-level protected mode~~ but when an address immediately prior to each one of the critical code segments in the execution flow is encountered, an exception occurs and the kernel-level execution mode is entered whereby control is transferred to the one or more exception handlers for execution of the critical code segments, and when the exception handlers complete execution, control is returned to the non-critical code segments.

37 (Currently Amended) The method of claim 36 wherein the providing (c) further includes: using the driver to initiate a set-up of debug registers, such that the critical code segments in the exception handlers are executed at appropriate times during ~~the program~~ execution of the computer program.

38 (Previously Presented) The method of claim 37 further including: executing the driver prior to a point in the program flow where any of the critical code segments was removed for placement into the exception handlers.

39 (Currently Amended) The method of claim 37 wherein the providing (b) further includes: inserting each of the exception handlers into ~~the~~ a linked list.

40 (Currently Amended) The method of claim 39 further includes: when one of the exceptions is raised to the operating system, handing control down the linked list until one of the exception handlers determines that the exception is one that the exception handler is designed to handle.

41 (Previously Presented) The method of claim 40 further includes: if the exception is one for which the current exception handler was designed to handle, executing the critical code segment included in the current exception handler.

42 (Previously Presented) The method of claim 41 further including: if the exception is one for which the exception set-up handler was designed to handle, setting a return address for the exception set-up handler when the exception processing completes.

43 (Previously Presented) The method of claim 42 further including: modifying debug registers during execution of the exception handlers such that any number of exception handlers may be daisy chained.

44 (Currently Amended) The method of claim 36 wherein the providing (b) further includes providing entry code for the ~~first~~ exception handlers to determine a nature of the exception.

45 (Currently Amended) The method of claim 44, further comprising handling the exception if the nature of the exception is applicable to ~~the~~ a first exception handler, otherwise handing exception processing to the next available exception handler.

46 (Currently Amended) The method of claim 36, further comprising selecting from the ~~plurality of~~one or more code segments the segments to obfuscate within the ~~first~~ exception handlers based on ~~the~~ a value of obfuscating the algorithms within the segment.

47 (Currently Amended) The method of claim 42, further comprising providing code for the ~~first~~ exception handlers to modify the return address to be used after completion of the exception processing

48 (Currently Amended) The method of claim 47, further comprising rearranging the ~~remaining~~ non-~~embedded~~ critical code segments out of normal execution order, and setting exception addresses and return addresses to ensure proper program sequencing to further obfuscate the program execution flow.

49 (Currently Amended) The method of claim 42, further comprising including code within the ~~first~~ exception handlers to modify ~~the~~ exception conditions to support future exceptions.

50 (Currently Amended) The method of claim 49, further comprising including a plurality of critical code segments within at least one of the ~~first~~ exception handlers.

51 (Currently Amended) The method of claim 50, further comprising selecting which of a plurality of critical code segments within the at least one~~first~~ exception handler should be executed based on ~~the~~ exception information

52 (Previously Presented) The method of claim 36, further comprising including additional, unrelated driver code within the driver to further obfuscate the operation of the program execution flow.

53 (Currently Amended) The method of claim 42, further comprising ~~setting up~~embedding one or more exceptions ~~to occur~~ within one or more ~~first~~ other exception handlers, ~~to be handled by other first exception handlers,~~ to further obfuscate the program execution flow.

54 (Currently Amended) The method of claim 53, further comprising repeating the embedded exceptions within exception handlers such that ~~the~~ required exception processing occurs at a plurality of exception processing levels, to further obfuscate the program execution flow.

55 (Cancelled).

56 (Cancelled).

57 (Cancelled).

58 (Cancelled).

59 (Cancelled).

60 (Cancelled).

61 (Cancelled).

62 (Cancelled).

63 (Cancelled).

64 (Cancelled).

65 (Cancelled).

66 (Cancelled).

67 (Cancelled).

68 (Cancelled).

69 (Currently Amended) A system for obfuscation of program execution flow, comprising:

a computer system including a processor, memory, an operating system that provides kernel-level and user-level execution modes, and debug resources to support the generation and processing of exceptions at specified addresses; and

an obfuscated program comprising~~,~~:

~~a plurality of~~one or more non-critical code segments~~, such that when the computer system executes the program, the non-critical code segments are executed in the user-level execution mode, wherein an execution path of the non-critical code segments is visible to a debugger program executing in the user-level execution mode~~,

a plurality of codes segments determined to be critical code segments that have been removed and encapsulated within ~~respective~~ one or more exception handlers, and

a set-up handler for setting up the debug resources such that when the computer system executes the program,~~ the exception handlers containing the critical code segments are executed in the kernel level mode, wherein an execution path of the critical code~~

~~segments is hidden from the debugger program executing in the user-level execution mode~~ execution flow of program is maintained with the non-critical code segments executing in the user-level protected mode, but when an address immediately prior to each one of the critical code segments in the execution flow is encountered, an exception occurs and the kernel-level execution mode is entered whereby control is transferred to the one or more exception handlers for execution of the critical code segments, and when the exception handlers complete execution, control is returned to the non-critical code segments.

70 (New) A method for obfuscation of computer program execution flow to increase computer program security, the method comprising:

(a) partitioning the computer program into one or more non-critical code segments, and one or more critical code segments to be obfuscated;

(b) removing from the computer program, at least one of the critical code segments and embedding the removed critical code segments within one or more exception handlers, wherein the exception handlers run under kernel-level execution mode of a computer's operating system;

(c) adding to the computer program, the exception handlers and code for setting-up the exception handlers during the execution of the program;

(d) adding to the computer program, code for invoking at least one of the exception handlers at each location in the computer program where the critical code segments were to be invoked in the computer program prior to removal; and

(e) executing the computer program on the computer having an operating system that provides kernel-level and user-level execution modes, wherein execution flow of computer program is maintained where the non-critical code segments execute in the user-level protected mode, and where when each location in the computer program previously occupied by the critical code segments is encountered, an exception occurs and the kernel-level execution mode is entered whereby control is transferred to the one or more exception handlers for execution of the critical code segments, and when the exception handlers complete execution, control is returned to the non-critical code segments.